Network Working Group
Request for Comments: 1157
Obsoletes: RFC 1098

J. Case
SNMP Research
M. Fedor
Performance Systems International
M. Schoffstall
Performance Systems International
J. Davin
MIT Laboratory for Computer Science

May 1990

A Simple Network Management Protocol (SNMP)

Table of Contents

1. Status of this Memo

This RFC is a re-release of RFC 1098, with a changed "Status of this Memo" section plus a few minor typographical corrections. This memo defines a simple protocol by which management information for a network element may be inspected or altered by logically remote users. In particular, together with its companion memos which describe the structure of management information along with the management information base, these documents provide a simple, workable architecture and system for managing TCP/IP-based internets and in particular the Internet.

The Internet Activities Board recommends that all IP and TCP implementations be network manageable. This implies implementation of the Internet MIB (RFC-1156) and at least one of the two recommended management protocols SNMP (RFC-1157) or CMOT (RFC-1095). It should be noted that, at this time, SNMP is a full Internet standard and CMOT is a draft standard. See also the Host and Gateway Requirements RFCs for more specific information on the applicability of this standard.

Please refer to the latest edition of the "IAB Official Protocol Standards" RFC for current information on the state and status of standard Internet protocols.

Distribution of this memo is unlimited.

2. Introduction

As reported in RFC 1052, IAB Recommendations for the Development of Internet Network Management Standards [1], a two-prong strategy for network management of TCP/IP-based internets was undertaken. In the short-term, the Simple Network Management Protocol (SNMP) was to be used to manage nodes in the Internet community. In the long-term, the use of the OSI network management framework was to be examined. Two documents were produced to define the management information: RFC 1065, which defined the Structure of Management Information (SMI) [2], and RFC 1066, which defined the Management Information Base (MIB) [3]. Both of these documents were designed so as to be

compatible with both the SNMP and the OSI network management framework.

This strategy was quite successful in the short-term: Internet-based

network management technology was fielded, by both the research and commercial communities, within a few months. As a result of this, portions of the Internet community became network manageable in a timely fashion.

As reported in RFC 1109, Report of the Second Ad Hoc Network Management Review Group [4], the requirements of the SNMP and the OSI network management frameworks were more different than anticipated. As such, the requirement for compatibility between the SMI/MIB and both frameworks was suspended. This action permitted the operational network management framework, the SNMP, to respond to new operational needs in the Internet community by producing documents defining new MIB items.

The IAB has designated the SNMP, SMI, and the initial Internet MIB to be full "Standard Protocols" with "Recommended" status. By this action, the IAB recommends that all IP and TCP implementations be network manageable and that the implementations that are network manageable are expected to adopt and implement the SMI, MIB, and SNMP.

As such, the current network management framework for TCP/IP- based internets consists of: Structure and Identification of Management Information for TCP/IP-based Internets, which describes how managed objects contained in the MIB are defined as set forth in RFC 1155 [5]; Management Information Base for Network Management of TCP/IP-based Internets, which describes the managed objects contained in the MIB as set forth in RFC 1156 [6]; and, the Simple Network Management Protocol, which defines the protocol used to manage these objects, as set forth in this memo.

As reported in RFC 1052, IAB Recommendations for the Development of Internet Network Management Standards [1], the Internet Activities Board has directed the Internet Engineering Task Force (IETF) to create two new working groups in the area of network management. One group was charged with the further specification and definition of elements to be included in the Management Information Base (MIB). The other was charged with defining the modifications to the Simple Network Management Protocol (SNMP) to accommodate the short-term needs of the network vendor and operations communities, and to align with the output of the MIB working group.

The MIB working group produced two memos, one which defines a Structure for Management Information (SMI) [2] for use by the managed

objects contained in the MIB. A second memo [3] defines the list of managed objects.

The output of the SNMP Extensions working group is this memo, which incorporates changes to the initial SNMP definition [7] required to attain alignment with the output of the MIB working group. The changes should be minimal in order to be consistent with the IAB's directive that the working groups be "extremely sensitive to the need

to keep the SNMP simple." Although considerable care and debate has gone into the changes to the SNMP which are reflected in this memo, the resulting protocol is not backwardly-compatible with its predecessor, the Simple Gateway Monitoring Protocol (SGMP) [8]. Although the syntax of the protocol has been altered, the original philosophy, design decisions, and architecture remain intact. In order to avoid confusion, new UDP ports have been allocated for use by the protocol described in this memo.

3. The SNMP Architecture

Implicit in the SNMP architectural model is a collection of network management stations and network elements. Network management stations execute management applications which monitor and control network elements. Network elements are devices such as hosts, gateways, terminal servers, and the like, which have management agents responsible for performing the network management functions requested by the network management stations. The Simple Network Management Protocol (SNMP) is used to communicate management information between the network management stations and the agents in the network elements.

3.1. Goals of the Architecture

The SNMP explicitly minimizes the number and complexity of management functions realized by the management agent itself. This goal is attractive in at least four respects:

(1) The development cost for management agent software necessary to support the protocol is accordingly reduced.

(2) The degree of management function that is remotely supported is accordingly increased, thereby admitting fullest use of internet resources in the management task.

(3) The degree of management function that is remotely supported is accordingly increased, thereby imposing the fewest possible restrictions on the form and sophistication of management tools.

(4) Simplified sets of management functions are easily understood and used by developers of network management tools.

A second goal of the protocol is that the functional paradigm for monitoring and control be sufficiently extensible to accommodate additional, possibly unanticipated aspects of network operation and management.

A third goal is that the architecture be, as much as possible, independent of the architecture and mechanisms of particular hosts or particular gateways.

3.2. Elements of the Architecture

The SNMP architecture articulates a solution to the network management problem in terms of:

(1) the scope of the management information communicated by the protocol,

(2) the representation of the management information communicated by the protocol,

(3) operations on management information supported by the protocol,

(4) the form and meaning of exchanges among management entities,

(5) the definition of administrative relationships among management entities, and

(6) the form and meaning of references to management

information.

3.2.1. Scope of Management Information

The scope of the management information communicated by operation of the SNMP is exactly that represented by instances of all non-aggregate object types either defined in Internet-standard MIB or defined elsewhere according to the conventions set forth in Internet-standard SMI [5].

Support for aggregate object types in the MIB is neither required for conformance with the SMI nor realized by the SNMP.

3.2.2. Representation of Management Information

Management information communicated by operation of the SNMP is represented according to the subset of the ASN.1 language [9] that is specified for the definition of non-aggregate types in the SMI.

The SGMP adopted the convention of using a well-defined subset of the ASN.1 language [9]. The SNMP continues and extends this tradition by utilizing a moderately more complex subset of ASN.1 for describing managed objects and for describing the protocol data units used for managing those objects. In addition, the desire to ease eventual transition to OSI-based network management protocols led to the definition in the ASN.1 language of an Internet-standard Structure of Management Information (SMI) [5] and Management Information Base (MIB) [6]. The use of the ASN.1 language, was, in part, encouraged by the successful use of ASN.1 in earlier efforts, in particular, the SGMP. The restrictions on the use of ASN.1 that are part of the SMI contribute to the simplicity espoused and validated by experience with the SGMP.

Also for the sake of simplicity, the SNMP uses only a subset of the basic encoding rules of ASN.1 [10]. Namely, all encodings use the definite-length form. Further, whenever permissible, non-constructor encodings are used rather than constructor encodings. This restriction applies to all aspects of ASN.1 encoding, both for the top-level protocol data units and the data objects they contain.

3.2.3. Operations Supported on Management Information

The SNMP models all management agent functions as alterations or inspections of variables. Thus, a protocol entity on a logically remote host (possibly the network element itself) interacts with the management agent resident on the network element in order to retrieve (get) or alter (set) variables. This strategy has at least two positive consequences:

(1) It has the effect of limiting the number of essential management functions realized by the management agent to two: one operation to assign a value to a specified configuration or other parameter and another to retrieve

such a value.

(2) A second effect of this decision is to avoid introducing into the protocol definition support for imperative management commands: the number of such commands is in practice ever-increasing, and the semantics of such commands are in general arbitrarily complex.

The strategy implicit in the SNMP is that the monitoring of network state at any significant level of detail is accomplished primarily by polling for appropriate information on the part of the monitoring center(s). A limited number of unsolicited messages (traps) guide the timing and focus of the polling. Limiting the number of unsolicited messages is consistent with the goal of simplicity and minimizing the amount of traffic generated by the network management function.

The exclusion of imperative commands from the set of explicitly supported management functions is unlikely to preclude any desirable management agent operation. Currently, most commands are requests either to set the value of some parameter or to retrieve such a value, and the function of the few imperative commands currently supported is easily accommodated in an asynchronous mode by this management model. In this scheme, an imperative command might be realized as the setting of a parameter value that subsequently triggers the desired action. For example, rather than implementing a "reboot command," this action might be invoked by simply setting a parameter indicating the number of seconds until system reboot.

3.2.4. Form and Meaning of Protocol Exchanges

The communication of management information among management entities is realized in the SNMP through the exchange of protocol messages. The form and meaning of those messages is defined below in Section 4.

Consistent with the goal of minimizing complexity of the management agent, the exchange of SNMP messages requires only an unreliable datagram service, and every message is entirely and independently represented by a single transport datagram. While this document specifies the exchange of messages via the UDP protocol [11], the mechanisms of the SNMP are generally suitable for use with a wide variety of transport services.

3.2.5. Definition of Administrative Relationships

The SNMP architecture admits a variety of administrative relationships among entities that participate in the protocol. The entities residing at management stations and network elements which communicate with one another using the SNMP are termed SNMP application entities. The peer processes which implement the SNMP, and thus support the SNMP application entities, are termed protocol entities.

A pairing of an SNMP agent with some arbitrary set of SNMP application entities is called an SNMP community. Each SNMP community is named by a string of octets, that is called the community name for said community.

An SNMP message originated by an SNMP application entity that in fact belongs to the SNMP community named by the community component of said message is called an authentic SNMP message. The set of rules by which an SNMP message is identified as an authentic SNMP message for a particular SNMP community is called an authentication scheme. An implementation of a function that identifies authentic SNMP messages according to one or more authentication schemes is called an authentication service.

Clearly, effective management of administrative relationships among SNMP application entities requires authentication services that (by the use of encryption or other techniques) are able to identify authentic SNMP messages with a high degree of certainty. Some SNMP implementations may wish to support only a trivial authentication service that identifies all SNMP messages as authentic SNMP messages.

For any network element, a subset of objects in the MIB that pertain to that element is called a SNMP MIB view. Note that the names of the object types represented in a SNMP MIB view need not belong to a

single sub-tree of the object type name space.

An element of the set { READ-ONLY, READ-WRITE } is called an SNMP access mode.

A pairing of a SNMP access mode with a SNMP MIB view is called an SNMP community profile. A SNMP community profile represents specified access privileges to variables in a specified MIB view. For every variable in the MIB view in a given SNMP community profile, access to that variable is represented by the profile according to the following conventions:

(1) if said variable is defined in the MIB with "Access:" of "none," it is unavailable as an operand for any operator;

(2) if said variable is defined in the MIB with "Access:" of "read-write" or "write-only" and the access mode of the given profile is READ-WRITE, that variable is available as an operand for the get, set, and trap operations;

(3) otherwise, the variable is available as an operand for the get and trap operations.

(4) In those cases where a "write-only" variable is an operand used for the get or trap operations, the value given for the variable is implementation-specific.

A pairing of a SNMP community with a SNMP community profile is called

a SNMP access policy. An access policy represents a specified community profile afforded by the SNMP agent of a specified SNMP community to other members of that community. All administrative relationships among SNMP application entities are architecturally defined in terms of SNMP access policies.

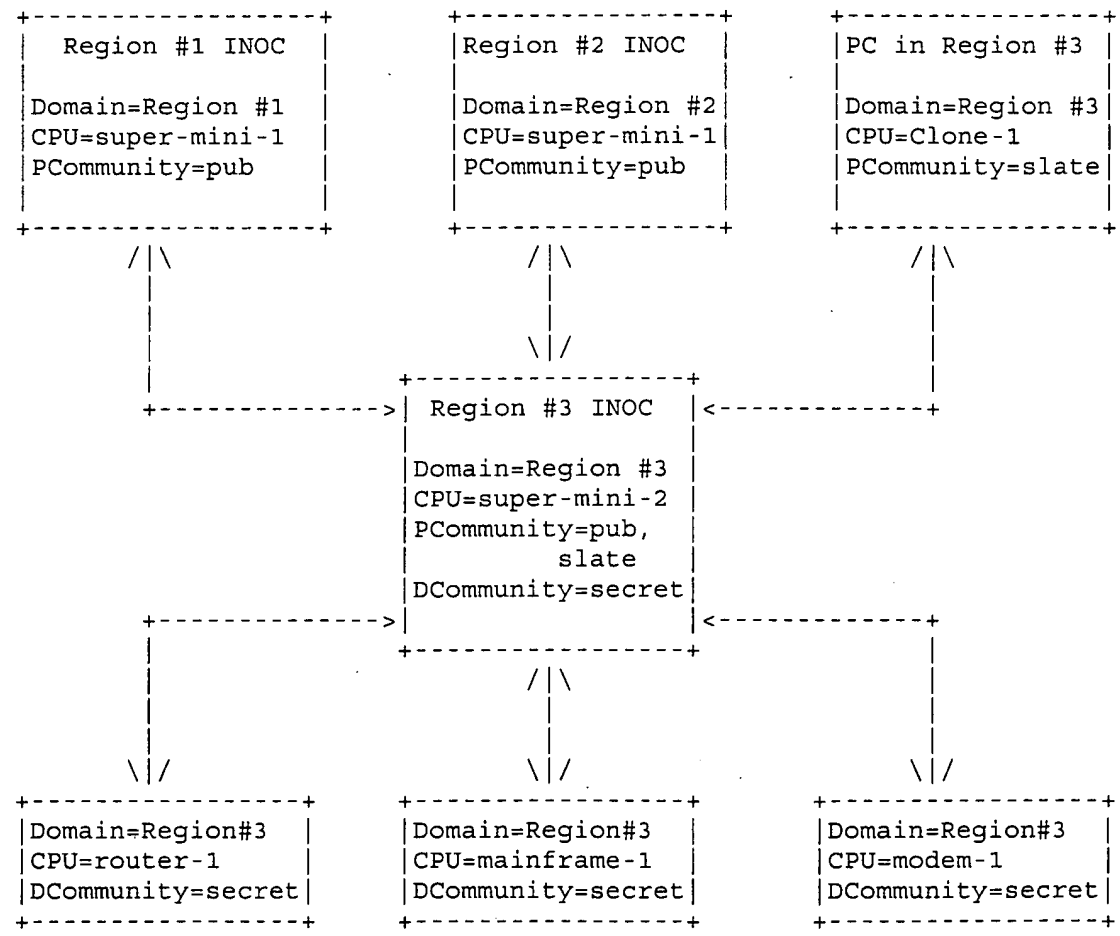
For every SNMP access policy, if the network element on which the SNMP agent for the specified SNMP community resides is not that to which the MIB view for the specified profile pertains, then that policy is called a SNMP proxy access policy. The SNMP agent associated with a proxy access policy is called a SNMP proxy agent. While careless definition of proxy access policies can result in management loops, prudent definition of proxy policies is useful in at least two ways:

(1) It permits the monitoring and control of network elements which are otherwise not addressable using the management protocol and the transport protocol. That is, a proxy agent may provide a protocol conversion function allowing a management station to apply a consistent management

framework to all network elements, including devices such as modems, multiplexors, and other devices which support different management frameworks.

(2) It potentially shields network elements from elaborate access control policies. For example, a proxy agent may implement sophisticated access control whereby diverse subsets of variables within the MIB are made accessible to different management stations without increasing the complexity of the network element.

By way of example, Figure 1 illustrates the relationship between management stations, proxy agents, and management agents. In this example, the proxy agent is envisioned to be a normal Internet Network Operations Center (INOC) of some administrative domain which has a standard managerial relationship with a set of management agents.



Domain: the administrative domain of the element
PCommunity: the name of a community utilizing a proxy agent
DCommunity: the name of a direct community

Figure 1
Example Network Management Configuration

3.2.6. Form and Meaning of References to Managed Objects

The SMI requires that the definition of a conformant management protocol address:

(1) the resolution of ambiguous MIB references,

(2) the resolution of MIB references in the presence multiple MIB versions, and

(3) the identification of particular instances of object types defined in the MIB.

3.2.6.1. Resolution of Ambiguous MIB References

Because the scope of any SNMP operation is conceptually confined to objects relevant to a single network element, and because all SNMP references to MIB objects are (implicitly or explicitly) by unique variable names, there is no possibility that any SNMP reference to any object type defined in the MIB could resolve to multiple instances of that type.

3.2.6.2. Resolution of References across MIB Versions

The object instance referred to by any SNMP operation is exactly that specified as part of the operation request or (in the case of a get-next operation) its immediate successor in the MIB as a whole. In particular, a reference to an object as part of some version of the Internet-standard MIB does not resolve to any object that is not part of said version of the Internet-standard MIB, except in the case that the requested operation is get-next and the specified object name is lexicographically last among the names of all objects presented as part of said version of the Internet-Standard MIB.

3.2.6.3. Identification of Object Instances

The names for all object types in the MIB are defined explicitly either in the Internet-standard MIB or in other documents which conform to the naming conventions of the SMI. The SMI requires that conformant management protocols define mechanisms for identifying

individual instances of those object types for a particular network element.

Each instance of any object type defined in the MIB is identified in SNMP operations by a unique name called its "variable name." In general, the name of an SNMP variable is an OBJECT IDENTIFIER of the form x.y, where x is the name of a non-aggregate object type defined in the MIB and y is an OBJECT IDENTIFIER fragment that, in a way

specific to the named object type, identifies the desired instance.

This naming strategy admits the fullest exploitation of the semantics of the GetNextRequest-PDU (see Section 4), because it assigns names for related variables so as to be contiguous in the lexicographical ordering of all variable names known in the MIB.

The type-specific naming of object instances is defined below for a number of classes of object types. Instances of an object type to which none of the following naming conventions are applicable are named by OBJECT IDENTIFIERs of the form x.0, where x is the name of said object type in the MIB definition.

For example, suppose one wanted to identify an instance of the variable sysDescr The object class for sysDescr is:

```
iso org dod internet mgmt mib system sysDescr
 1   3   6     1      2    1    1       1
```

Hence, the object type, x, would be 1.3.6.1.2.1.1.1 to which is appended an instance sub-identifier of 0. That is, 1.3.6.1.2.1.1.1.0 identifies the one and only instance of sysDescr.

3.2.6.3.1. ifTable Object Type Names

The name of a subnet interface, s, is the OBJECT IDENTIFIER value of the form i, where i has the value of that instance of the ifIndex object type associated with s.

For each object type, t, for which the defined name, n, has a prefix of ifEntry, an instance, i, of t is named by an OBJECT IDENTIFIER of the form n.s, where s is the name of the subnet interface about which i represents information.

For example, suppose one wanted to identify the instance of the variable ifType associated with interface 2. Accordingly, ifType.2 would identify the desired instance.

3.2.6.3.2. atTable Object Type Names

The name of an AT-cached network address, x, is an OBJECT IDENTIFIER of the form 1.a.b.c.d, where a.b.c.d is the value (in the familiar "dot" notation) of the atNetAddress object type associated with x.

The name of an address translation equivalence e is an OBJECT IDENTIFIER value of the form s.w, such that s is the value of that instance of the atIndex object type associated with e and such that w is the name of the AT-cached network address associated with e.

For each object type, t, for which the defined name, n, has a prefix of atEntry, an instance, i, of t is named by an OBJECT IDENTIFIER of the form n.y, where y is the name of the address translation equivalence about which i represents information.

For example, suppose one wanted to find the physical address of an entry in the address translation table (ARP cache) associated with an IP address of 89.1.1.42 and interface 3. Accordingly, atPhysAddress.3.1.89.1.1.42 would identify the desired instance.

3.2.6.3.3. ipAddrTable Object Type Names

The name of an IP-addressable network element, x, is the OBJECT IDENTIFIER of the form a.b.c.d such that a.b.c.d is the value (in the familiar "dot" notation) of that instance of the ipAdEntAddr object type associated with x.

For each object type, t, for which the defined name, n, has a prefix of ipAddrEntry, an instance, i, of t is named by an OBJECT IDENTIFIER of the form n.y, where y is the name of the IP-addressable network element about which i represents information.

For example, suppose one wanted to find the network mask of an entry in the IP interface table associated with an IP address of 89.1.1.42. Accordingly, ipAdEntNetMask.89.1.1.42 would identify the desired instance.

3.2.6.3.4. ipRoutingTable Object Type Names

The name of an IP route, x, is the OBJECT IDENTIFIER of the form a.b.c.d such that a.b.c.d is the value (in the familiar "dot" notation) of that instance of the ipRouteDest object type associated with x.

For each object type, t, for which the defined name, n, has a prefix of ipRoutingEntry, an instance, i, of t is named by an OBJECT IDENTIFIER of the form n.y, where y is the name of the IP route about which i represents information.

For example, suppose one wanted to find the next hop of an entry in the IP routing table associated with the destination of 89.1.1.42. Accordingly, ipRouteNextHop.89.1.1.42 would identify the desired instance.

3.2.6.3.5. tcpConnTable Object Type Names

The name of a TCP connection, x, is the OBJECT IDENTIFIER of the form a.b.c.d.e.f.g.h.i.j such that a.b.c.d is the value (in the familiar

"dot" notation) of that instance of the tcpConnLocalAddress object type associated with x and such that f.g.h.i is the value (in the familiar "dot" notation) of that instance of the tcpConnRemoteAddress object type associated with x and such that e is the value of that instance of the tcpConnLocalPort object type associated with x and such that j is the value of that instance of the tcpConnRemotePort object type associated with x.

For each object type, t, for which the defined name, n, has a prefix of tcpConnEntry, an instance, i, of t is named by an OBJECT IDENTIFIER of the form n.y, where y is the name of the TCP connection about which i represents information.

For example, suppose one wanted to find the state of a TCP connection between the local address of 89.1.1.42 on TCP port 21 and the remote address of 10.0.0.51 on TCP port 2059. Accordingly, tcpConnState.89.1.1.42.21.10.0.0.51.2059 would identify the desired instance.

3.2.6.3.6. egpNeighTable Object Type Names

The name of an EGP neighbor, x, is the OBJECT IDENTIFIER of the form a.b.c.d such that a.b.c.d is the value (in the familiar "dot" notation) of that instance of the egpNeighAddr object type associated with x.

For each object type, t, for which the defined name, n, has a prefix of egpNeighEntry, an instance, i, of t is named by an OBJECT IDENTIFIER of the form n.y, where y is the name of the EGP neighbor about which i represents information.

For example, suppose one wanted to find the neighbor state for the IP address of 89.1.1.42. Accordingly, egpNeighState.89.1.1.42 would identify the desired instance.

4. Protocol Specification

The network management protocol is an application protocol by which the variables of an agent's MIB may be inspected or altered.

Communication among protocol entities is accomplished by the exchange of messages, each of which is entirely and independently represented within a single UDP datagram using the basic encoding rules of ASN.1 (as discussed in Section 3.2.2). A message consists of a version identifier, an SNMP community name, and a protocol data unit (PDU). A protocol entity receives messages at UDP port 161 on the host with which it is associated for all messages except for those which report traps (i.e., all messages except those which contain the Trap-PDU). Messages which report traps should be received on UDP port 162 for further processing. An implementation of this protocol need not accept messages whose length exceeds 484 octets. However, it is recommended that implementations support larger datagrams whenever feasible.

It is mandatory that all implementations of the SNMP support the five PDUs: GetRequest-PDU, GetNextRequest-PDU, GetResponse-PDU, SetRequest-PDU, and Trap-PDU.

```
RFC1157-SNMP DEFINITIONS ::= BEGIN

 IMPORTS
      ObjectName, ObjectSyntax, NetworkAddress, IpAddress, TimeTicks
              FROM RFC1155-SMI;


 -- top-level message

         Message ::=
                 SEQUENCE {
                      version        -- version-1 for this RFC
                         INTEGER {
                             version-1(0)
                         },

                     community      -- community name
                         OCTET STRING,

                     data           -- e.g., PDUs if trivial
                         ANY        -- authentication is being used
                 }
```

```
-- protocol data units

        PDUs ::=
                CHOICE {
                    get-request
                        GetRequest-PDU,

                    get-next-request
                        GetNextRequest-PDU,

                    get-response
                        GetResponse-PDU,

                    set-request
                        SetRequest-PDU,

                    trap
                        Trap-PDU
                     }

-- the individual PDUs and commonly used
-- data types will be defined later

END
```

4.1. Elements of Procedure

This section describes the actions of a protocol entity implementing the SNMP. Note, however, that it is not intended to constrain the internal architecture of any conformant implementation.

In the text that follows, the term transport address is used. In the case of the UDP, a transport address consists of an IP address along with a UDP port. Other transport services may be used to support the SNMP. In these cases, the definition of a transport address should be made accordingly.

The top-level actions of a protocol entity which generates a message are as follows:

(1) It first constructs the appropriate PDU, e.g., the GetRequest-PDU, as an ASN.1 object.

(2) It then passes this ASN.1 object along with a community name its source transport address and the destination transport address, to the service which implements the desired authentication scheme. This authentication

service returns another ASN.1 object.

(3) The protocol entity then constructs an ASN.1 Message object, using the community name and the resulting ASN.1

object.

(4) This new ASN.1 object is then serialized, using the basic encoding rules of ASN.1, and then sent using a transport service to the peer protocol entity.

Similarly, the top-level actions of a protocol entity which receives a message are as follows:

(1) It performs a rudimentary parse of the incoming datagram to build an ASN.1 object corresponding to an ASN.1 Message object. If the parse fails, it discards the datagram and performs no further actions.

(2) It then verifies the version number of the SNMP message. If there is a mismatch, it discards the datagram and performs no further actions.

(3) The protocol entity then passes the community name and user data found in the ASN.1 Message object, along with the datagram's source and destination transport addresses to the service which implements the desired authentication scheme. This entity returns another ASN.1 object, or signals an authentication failure. In the latter case, the protocol entity notes this failure, (possibly) generates a trap, and discards the datagram and performs no further actions.

(4) The protocol entity then performs a rudimentary parse on the ASN.1 object returned from the authentication service to build an ASN.1 object corresponding to an ASN.1 PDUs object. If the parse fails, it discards the datagram and performs no further actions. Otherwise, using the named SNMP community, the appropriate profile is selected, and the PDU is processed accordingly. If, as a result of this processing, a message is returned then the source transport address that the response message is sent from shall be identical to the destination transport address that the original request message was sent to.

4.1.1. Common Constructs

Before introducing the six PDU types of the protocol, it is appropriate to consider some of the ASN.1 constructs used frequently:

```
-- request/response information

RequestID ::=
```

```
            INTEGER

        ErrorStatus ::=
            INTEGER {
                noError(0),
                tooBig(1),
                noSuchName(2),
                badValue(3),
                readOnly(4)
                genErr(5)
            }

        ErrorIndex ::=
            INTEGER


        -- variable bindings

        VarBind ::=
            SEQUENCE {
                name
                    ObjectName,

                value
                    ObjectSyntax
            }

        VarBindList ::=
            SEQUENCE OF
                VarBind
```

RequestIDs are used to distinguish among outstanding requests. By use of the RequestID, an SNMP application entity can correlate incoming responses with outstanding requests. In cases where an unreliable datagram service is being used, the RequestID also provides a simple means of identifying messages duplicated by the network.

A non-zero instance of ErrorStatus is used to indicate that an

exception occurred while processing a request. In these cases, ErrorIndex may provide additional information by indicating which variable in a list caused the exception.

The term variable refers to an instance of a managed object. A variable binding, or VarBind, refers to the pairing of the name of a variable to the variable's value. A VarBindList is a simple list of variable names and corresponding values. Some PDUs are concerned only with the name of a variable and not its value (e.g., the GetRequest-PDU). In this case, the value portion of the binding is ignored by the protocol entity. However, the value portion must still have valid ASN.1 syntax and encoding. It is recommended that

the ASN.1 value NULL be used for the value portion of such bindings.

4.1.2. The GetRequest-PDU

The form of the GetRequest-PDU is:

```
GetRequest-PDU ::=
    [0]
        IMPLICIT SEQUENCE {
            request-id
                RequestID,

            error-status        -- always 0
                ErrorStatus,

            error-index         -- always 0
                ErrorIndex,

            variable-bindings
                VarBindList
        }
```

The GetRequest-PDU is generated by a protocol entity only at the request of its SNMP application entity.

Upon receipt of the GetRequest-PDU, the receiving protocol entity responds according to any applicable rule in the list below:

(1) If, for any object named in the variable-bindings field, the object's name does not exactly match the name of some object available for get operations in the relevant MIB view, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is noSuchName, and the value of the error-index field is the index of said object name component in the received

message.

(2) If, for any object named in the variable-bindings field, the object is an aggregate type (as defined in the SMI), then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is noSuchName, and the value of the error-index field is the index of said object name component in the received message.

(3) If the size of the GetResponse-PDU generated as described below would exceed a local limitation, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is tooBig, and the value

of the error-index field is zero.

(4) If, for any object named in the variable-bindings field, the value of the object cannot be retrieved for reasons not covered by any of the foregoing rules, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is genErr and the value of the error-index field is the index of said object name component in the received message.

If none of the foregoing rules apply, then the receiving protocol entity sends to the originator of the received message the GetResponse-PDU such that, for each object named in the variable-bindings field of the received message, the corresponding component of the GetResponse-PDU represents the name and value of that variable. The value of the error- status field of the GetResponse-PDU is noError and the value of the error-index field is zero. The value of the request-id field of the GetResponse-PDU is that of the received message.

4.1.3. The GetNextRequest-PDU

The form of the GetNextRequest-PDU is identical to that of the GetRequest-PDU except for the indication of the PDU type. In the ASN.1 language:

```
GetNextRequest-PDU ::=
    [1]
        IMPLICIT SEQUENCE {
            request-id
                RequestID,
```



```
            error-status        -- always 0
                ErrorStatus,

            error-index         -- always 0
                ErrorIndex,

            variable-bindings
                VarBindList
        }
```

The GetNextRequest-PDU is generated by a protocol entity only at the request of its SNMP application entity.

Upon receipt of the GetNextRequest-PDU, the receiving protocol entity responds according to any applicable rule in the list below:

(1) If, for any object name in the variable-bindings field, that name does not lexicographically precede the name of some object available for get operations in the relevant

MIB view, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is noSuchName, and the value of the error-index field is the index of said object name component in the received message.

(2) If the size of the GetResponse-PDU generated as described below would exceed a local limitation, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is tooBig, and the value of the error-index field is zero.

(3) If, for any object named in the variable-bindings field, the value of the lexicographical successor to the named object cannot be retrieved for reasons not covered by any of the foregoing rules, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is genErr and the value of the error-index field is the index of said object name component in the received message.

If none of the foregoing rules apply, then the receiving protocol entity sends to the originator of the received message the GetResponse-PDU such that, for each name in the variable-bindings field of the received message, the corresponding component of the

GetResponse-PDU represents the name and value of that object whose name is, in the lexicographical ordering of the names of all objects available for get operations in the relevant MIB view, together with the value of the name field of the given component, the immediate successor to that value. The value of the error-status field of the GetResponse-PDU is noError and the value of the errorindex field is zero. The value of the request-id field of the GetResponse-PDU is that of the received message.

4.1.3.1. Example of Table Traversal

One important use of the GetNextRequest-PDU is the traversal of conceptual tables of information within the MIB. The semantics of this type of SNMP message, together with the protocol-specific mechanisms for identifying individual instances of object types in the MIB, affords access to related objects in the MIB as if they enjoyed a tabular organization.

By the SNMP exchange sketched below, an SNMP application entity might extract the destination address and next hop gateway for each entry in the routing table of a particular network element. Suppose that this routing table has three entries:

Destination	NextHop	Metric

```
10.0.0.99                       89.1.1.42       5
9.1.2.3                         99.0.0.3        3
10.0.0.51                       89.1.1.42       5
```

The management station sends to the SNMP agent a GetNextRequest-PDU containing the indicated OBJECT IDENTIFIER values as the requested variable names:

```
GetNextRequest ( ipRouteDest, ipRouteNextHop, ipRouteMetric1 )
```

The SNMP agent responds with a GetResponse-PDU:

```
GetResponse (( ipRouteDest.9.1.2.3 =  "9.1.2.3" ),
        ( ipRouteNextHop.9.1.2.3 = "99.0.0.3" ),
        ( ipRouteMetric1.9.1.2.3 = 3 ))
```

The management station continues with:

```
GetNextRequest ( ipRouteDest.9.1.2.3,
        ipRouteNextHop.9.1.2.3,
```



```
        ipRouteMetric1.9.1.2.3 )
```

The SNMP agent responds:

```
GetResponse (( ipRouteDest.10.0.0.51 = "10.0.0.51" ),
        ( ipRouteNextHop.10.0.0.51 = "89.1.1.42" ),
        ( ipRouteMetric1.10.0.0.51 = 5 ))
```

The management station continues with:

```
GetNextRequest ( ipRouteDest.10.0.0.51,
        ipRouteNextHop.10.0.0.51,
        ipRouteMetric1.10.0.0.51 )
```

The SNMP agent responds:

```
GetResponse (( ipRouteDest.10.0.0.99 = "10.0.0.99" ),
        ( ipRouteNextHop.10.0.0.99 = "89.1.1.42" ),
        ( ipRouteMetric1.10.0.0.99 = 5 ))
```

The management station continues with:

```
GetNextRequest ( ipRouteDest.10.0.0.99,
        ipRouteNextHop.10.0.0.99,
```

```
                    ipRouteMetric1.10.0.0.99 )
```

As there are no further entries in the table, the SNMP agent returns those objects that are next in the lexicographical ordering of the known object names. This response signals the end of the routing table to the management station.

4.1.4. The GetResponse-PDU

The form of the GetResponse-PDU is identical to that of the GetRequest-PDU except for the indication of the PDU type. In the ASN.1 language:

```
GetResponse-PDU ::=
    [2]
        IMPLICIT SEQUENCE {
            request-id
                RequestID,
```



```
            error-status
                ErrorStatus,

            error-index
                ErrorIndex,

            variable-bindings
                VarBindList
        }
```

The GetResponse-PDU is generated by a protocol entity only upon receipt of the GetRequest-PDU, GetNextRequest-PDU, or SetRequest-PDU, as described elsewhere in this document.

Upon receipt of the GetResponse-PDU, the receiving protocol entity presents its contents to its SNMP application entity.

4.1.5. The SetRequest-PDU

The form of the SetRequest-PDU is identical to that of the GetRequest-PDU except for the indication of the PDU type. In the ASN.1 language:

```
SetRequest-PDU ::=
    [3]
        IMPLICIT SEQUENCE {
            request-id
                RequestID,

            error-status        -- always 0
                ErrorStatus,
```

```
            error-index         -- always 0
                ErrorIndex,

            variable-bindings
                VarBindList
        }
```

The SetRequest-PDU is generated by a protocol entity only at the request of its SNMP application entity.

Upon receipt of the SetRequest-PDU, the receiving entity responds according to any applicable rule in the list below:

(1) If, for any object named in the variable-bindings field,

the object is not available for set operations in the relevant MIB view, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is noSuchName, and the value of the error-index field is the index of said object name component in the received message.

(2) If, for any object named in the variable-bindings field, the contents of the value field does not, according to the ASN.1 language, manifest a type, length, and value that is consistent with that required for the variable, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is badValue, and the value of the error-index field is the index of said object name in the received message.

(3) If the size of the Get Response type message generated as described below would exceed a local limitation, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is tooBig, and the value of the error-index field is zero.

(4) If, for any object named in the variable-bindings field, the value of the named object cannot be altered for reasons not covered by any of the foregoing rules, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is genErr and the value of the error-index field is the index of said object name component in the received message.

If none of the foregoing rules apply, then for each object named in the variable-bindings field of the received message, the

corresponding value is assigned to the variable. Each variable assignment specified by the SetRequest-PDU should be effected as if simultaneously set with respect to all other assignments specified in the same message.

The receiving entity then sends to the originator of the received message the GetResponse-PDU of identical form except that the value of the error-status field of the generated message is noError and the value of the error-index field is zero.

4.1.6. The Trap-PDU

The form of the Trap-PDU is:

```
  Trap-PDU ::=
      [4]

          IMPLICIT SEQUENCE {
             enterprise          -- type of object generating
                                 -- trap, see sysObjectID in [5]
                 OBJECT IDENTIFIER,

             agent-addr          -- address of object generating
                 NetworkAddress, -- trap

             generic-trap        -- generic trap type
                 INTEGER {
                     coldStart(0),
                     warmStart(1),
                     linkDown(2),
                     linkUp(3),
                     authenticationFailure(4),
                     egpNeighborLoss(5),
                     enterpriseSpecific(6)
                 },

             specific-trap     -- specific code, present even
                 INTEGER,      -- if generic-trap is not
                               -- enterpriseSpecific

             time-stamp        -- time elapsed between the last
               TimeTicks,      -- (re)initialization of the network
                               -- entity and the generation of the
                                  trap

             variable-bindings   -- "interesting" information
                  VarBindList
          }
```

The Trap-PDU is generated by a protocol entity only at the request of the SNMP application entity. The means by which an SNMP application entity selects the destination addresses of the SNMP application entities is implementation-specific.

Upon receipt of the Trap-PDU, the receiving protocol entity presents its contents to its SNMP application entity.

The significance of the variable-bindings component of the Trap-PDU is implementation-specific.

Interpretations of the value of the generic-trap field are:

4.1.6.1. The coldStart Trap

A coldStart(0) trap signifies that the sending protocol entity is reinitializing itself such that the agent's configuration or the protocol entity implementation may be altered.

4.1.6.2. The warmStart Trap

A warmStart(1) trap signifies that the sending protocol entity is reinitializing itself such that neither the agent configuration nor the protocol entity implementation is altered.

4.1.6.3. The linkDown Trap

A linkDown(2) trap signifies that the sending protocol entity recognizes a failure in one of the communication links represented in the agent's configuration.

The Trap-PDU of type linkDown contains as the first element of its variable-bindings, the name and value of the ifIndex instance for the affected interface.

4.1.6.4. The linkUp Trap

A linkUp(3) trap signifies that the sending protocol entity recognizes that one of the communication links represented in the agent's configuration has come up.

The Trap-PDU of type linkUp contains as the first element of its variable-bindings, the name and value of the ifIndex instance for the affected interface.

4.1.6.5. The authenticationFailure Trap

An authenticationFailure(4) trap signifies that the sending protocol entity is the addressee of a protocol message that is not properly authenticated. While implementations of the SNMP must be capable of generating this trap, they must also be capable of suppressing the emission of such traps via an implementation-specific mechanism.

4.1.6.6. The egpNeighborLoss Trap

An egpNeighborLoss(5) trap signifies that an EGP neighbor for whom

the sending protocol entity was an EGP peer has been marked down and the peer relationship no longer obtains.

The Trap-PDU of type egpNeighborLoss contains as the first element of its variable-bindings, the name and value of the egpNeighAddr instance for the affected neighbor.

4.1.6.7. The enterpriseSpecific Trap

A enterpriseSpecific(6) trap signifies that the sending protocol entity recognizes that some enterprise-specific event has occurred. The specific-trap field identifies the particular trap which occurred.

5. Definitions

```
     RFC1157-SNMP DEFINITIONS ::= BEGIN

      IMPORTS
          ObjectName, ObjectSyntax, NetworkAddress, IpAddress, TimeTicks
              FROM RFC1155-SMI;


          -- top-level message

          Message ::=
                  SEQUENCE {
                      version           -- version-1 for this RFC
                          INTEGER {
                              version-1(0)
                          },

                      community         -- community name
                          OCTET STRING,

                      data              -- e.g., PDUs if trivial
                          ANY           -- authentication is being used
                  }


          -- protocol data units

          PDUs ::=
                  CHOICE {
                              get-request
                                  GetRequest-PDU,

                              get-next-request
                                  GetNextRequest-PDU,

                              get-response
                                  GetResponse-PDU,

                              set-request
                                  SetRequest-PDU,

                              trap
                                  Trap-PDU
                          }
```

```
-- PDUs

GetRequest-PDU ::=
    [0]
        IMPLICIT PDU

GetNextRequest-PDU ::=
    [1]
        IMPLICIT PDU

GetResponse-PDU ::=
    [2]
        IMPLICIT PDU

SetRequest-PDU ::=
    [3]
        IMPLICIT PDU

PDU ::=
        SEQUENCE {
           request-id
                INTEGER,

            error-status      -- sometimes ignored
                INTEGER {
                    noError(0),
                    tooBig(1),
                    noSuchName(2),
                    badValue(3),
                    readOnly(4),
                    genErr(5)
                },

            error-index       -- sometimes ignored
               INTEGER,

            variable-bindings -- values are sometimes ignored
                VarBindList
        }

Trap-PDU ::=
    [4]
       IMPLICIT SEQUENCE {
            enterprise        -- type of object generating
                              -- trap, see sysObjectID in [5]


                OBJECT IDENTIFIER,
```


RFC 1157 SNMP May 1990

```
                agent-addr        -- address of object generating
                    NetworkAddress, -- trap

                generic-trap      -- generic trap type
                    INTEGER {
                        coldStart(0),
                        warmStart(1),
                        linkDown(2),
                        linkUp(3),
                        authenticationFailure(4),
                        egpNeighborLoss(5),
                        enterpriseSpecific(6)
                    },

                specific-trap  -- specific code, present even
                    INTEGER,   -- if generic-trap is not
                               -- enterpriseSpecific

                time-stamp     -- time elapsed between the last
                    TimeTicks, -- (re)initialization of the
                               network
                               -- entity and the generation of the
                               trap

                 variable-bindings -- "interesting" information
                    VarBindList
            }


        -- variable bindings

        VarBind ::=
                SEQUENCE {
                    name
                        ObjectName,

                    value
                        ObjectSyntax
                }

        VarBindList ::=
                SEQUENCE OF
                   VarBind

        END
```


6. Acknowledgements

This memo was influenced by the IETF SNMP Extensions working group:

Karl Auerbach, Epilogue Technology
K. Ramesh Babu, Excelan
Amatzia Ben-Artzi, 3Com/Bridge
Lawrence Besaw, Hewlett-Packard
Jeffrey D. Case, University of Tennessee at Knoxville
Anthony Chung, Sytek
James Davidson, The Wollongong Group
James R. Davin, MIT Laboratory for Computer Science
Mark S. Fedor, NYSERNet
Phill Gross, The MITRE Corporation
Satish Joshi, ACC
Dan Lynch, Advanced Computing Environments
Keith McCloghrie, The Wollongong Group
Marshall T. Rose, The Wollongong Group (chair)
Greg Satz, cisco
Martin Lee Schoffstall, Rensselaer Polytechnic Institute
Wengyik Yeong, NYSERNet

7. References

[1] Cerf, V., "IAB Recommendations for the Development of Internet Network Management Standards", RFC 1052, IAB, April 1988.

[2] Rose, M., and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based internets",

RFC 1065, TWG, August 1988.

[3] McCloghrie, K., and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets", RFC 1066, TWG, August 1988.

[4] Cerf, V., "Report of the Second Ad Hoc Network Management Review Group", RFC 1109, IAB, August 1989.

[5] Rose, M., and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets", RFC 1155, Performance Systems International and Hughes LAN Systems, May 1990.

[6] McCloghrie, K., and M. Rose, "Management Information Base for Network Management of TCP/IP-based Internets", RFC 1156, Hughes LAN Systems and Performance Systems International, May 1990.

[7] Case, J., M. Fedor, M. Schoffstall, and J. Davin, "A Simple Network Management Protocol", Internet Engineering Task Force working note, Network Information Center, SRI International, Menlo Park, California, March 1988.

[8] Davin, J., J. Case, M. Fedor, and M. Schoffstall, "A Simple Gateway Monitoring Protocol", RFC 1028, Proteon, University of Tennessee at Knoxville, Cornell University, and Rensselaer Polytechnic Institute, November 1987.

[9] Information processing systems - Open Systems Interconnection, "Specification of Abstract Syntax Notation One (ASN.1)", International Organization for Standardization, International Standard 8824, December 1987.

[10] Information processing systems - Open Systems Interconnection, "Specification of Basic Encoding Rules for Abstract Notation One (ASN.1)", International

Organization for Standardization, International Standard 8825, December 1987.

[11] Postel, J., "User Datagram Protocol", RFC 768, USC/Information Sciences Institute, November 1980.

Security Considerations

Security issues are not discussed in this memo.

Authors' Addresses

Jeffrey D. Case
SNMP Research
P.O. Box 8593
Knoxville, TN 37996-4800

Phone: (615) 573-1434

Email: case@CS.UTK.EDU

Mark Fedor
Performance Systems International
Rensselaer Technology Park
125 Jordan Road
Troy, NY 12180

Phone: (518) 283-8860

Email: fedor@patton.NYSER.NET

Martin Lee Schoffstall
Performance Systems International
Rensselaer Technology Park
165 Jordan Road
Troy, NY 12180

Phone: (518) 283-8860

Email: schoff@NISC.NYSER.NET

James R. Davin
MIT Laboratory for Computer Science, NE43-507
545 Technology Square
Cambridge, MA 02139

Phone: (617) 253-6020

EMail: jrd@ptt.lcs.mit.edu

RFC 768

J. Postel
ISI
28 August 1980

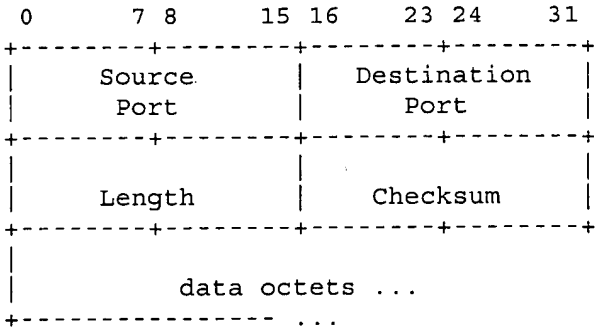
User Datagram Protocol

Introduction

This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

Format



User Datagram Header Format

Fields

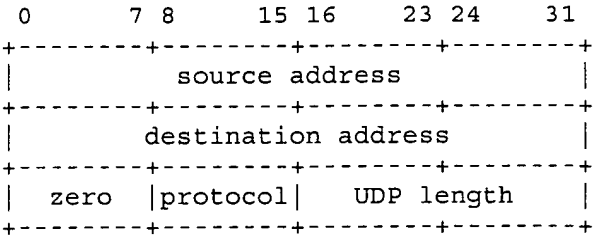
Source Port is an optional field, when meaningful, it indicates the port of the sending process, and may be assumed to be the port to which a reply should be addressed in the absence of any other information. If not used, a value of zero is inserted.

Destination Port has a meaning within the context of a particular internet destination address.

Length is the length in octets of this user datagram including this header and the data. (This means the minimum value of the length is eight.)

Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.

The pseudo header conceptually prefixed to the UDP header contains the source address, the destination address, the protocol, and the UDP length. This information gives protection against misrouted datagrams. This checksum procedure is the same as is used in TCP.



If the computed checksum is zero, it is transmitted as all ones (the equivalent in one's complement arithmetic). An all zero transmitted checksum value means that the transmitter generated no checksum (for debugging or for higher level protocols that don't care).

User Interface

A user interface should allow

the creation of new receive ports,

receive operations on the receive ports that return the data octets and an indication of source port and source address,

and an operation that allows a datagram to be sent, specifying the data, source and destination ports and addresses to be sent.

IP Interface

The UDP module must be able to determine the source and destination internet addresses and the protocol field from the internet header. One possible UDP/IP interface would return the whole internet datagram including all of the internet header in response to a receive operation. Such an interface would also allow the UDP to pass a full internet datagram complete with header to the IP to send. The IP would verify certain fields for consistency and compute the internet header checksum.

Protocol Application

The major uses of this protocol is the Internet Name Server [3], and the Trivial File Transfer [4].

Protocol Number

This is protocol 17 (21 octal) when used in the Internet Protocol. Other protocol numbers are listed in [5].

References
---

[1] Postel, J., "Internet Protocol," RFC 760, USC/Information Sciences Institute, January 1980.

[2] Postel, J., "Transmission Control Protocol," RFC 761, USC/Information Sciences Institute, January 1980.

[3] Postel, J., "Internet Name Server," USC/Information Sciences Institute, IEN 116, August 1979.

[4] Sollins, K., "The TFTP Protocol," Massachusetts Institute of Technology, IEN 133, January 1980.

[5] Postel, J., "Assigned Numbers," USC/Information Sciences Institute, RFC 762, January 1980.

Network Working Group
Request for Comments: 854

Obsoletes: NIC 18639

J. Postel
J. Reynolds
ISI

May 1983

TELNET PROTOCOL SPECIFICATION

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

INTRODUCTION

The purpose of the TELNET Protocol is to provide a fairly general, bi-directional, eight-bit byte oriented communications facility. Its primary goal is to allow a standard method of interfacing terminal devices and terminal-oriented processes to each other. It is envisioned that the protocol may also be used for terminal-terminal communication ("linking") and process-process communication (distributed computation).

GENERAL CONSIDERATIONS

A TELNET connection is a Transmission Control Protocol (TCP) connection used to transmit data with interspersed TELNET control information.

The TELNET Protocol is built upon three main ideas: first, the concept of a "Network Virtual Terminal"; second, the principle of negotiated options; and third, a symmetric view of terminals and processes.

1. When a TELNET connection is first established, each end is assumed to originate and terminate at a "Network Virtual Terminal", or NVT. An NVT is an imaginary device which provides a standard, network-wide, intermediate representation of a canonical terminal. This eliminates the need for "server" and "user" hosts to keep information about the characteristics of each other's terminals and terminal handling conventions. All hosts, both user and server, map their local device characteristics and conventions so as to appear to be dealing with an NVT over the network, and each can assume a similar mapping by the other party. The NVT is intended to strike a balance between being overly restricted (not providing hosts a rich enough vocabulary for mapping into their local character sets), and being overly inclusive (penalizing users with modest terminals).

NOTE: The "user" host is the host to which the physical terminal is normally attached, and the "server" host is the host which is normally providing some service. As an alternate point of view,

applicable even in terminal-to-terminal or process-to-process communications, the "user" host is the host which initiated the communication.

2. The principle of negotiated options takes cognizance of the fact that many hosts will wish to provide additional services over and above those available within an NVT, and many users will have sophisticated terminals and would like to have elegant, rather than minimal, services. Independent of, but structured within the TELNET Protocol are various "options" that will be sanctioned and may be used with the "DO, DON'T, WILL, WON'T" structure (discussed below) to allow a user and server to agree to use a more elaborate (or perhaps just different) set of conventions for their TELNET connection. Such options could include changing the character set, the echo mode, etc.

The basic strategy for setting up the use of options is to have either party (or both) initiate a request that some option take effect. The other party may then either accept or reject the request. If the request is accepted the option immediately takes effect; if it is rejected the associated aspect of the connection remains as specified for an NVT. Clearly, a party may always refuse a request to enable, and must never refuse a request to disable some option since all parties must be prepared to support the NVT.

The syntax of option negotiation has been set up so that if both parties request an option simultaneously, each will see the other's request as the positive acknowledgment of its own.

3. The symmetry of the negotiation syntax can potentially lead to nonterminating acknowledgment loops -- each party seeing the incoming commands not as acknowledgments but as new requests which must be acknowledged. To prevent such loops, the following rules prevail:

a. Parties may only request a change in option status; i.e., a party may not send out a "request" merely to announce what mode it is in.

b. If a party receives what appears to be a request to enter some mode it is already in, the request should not be acknowledged. This non-response is essential to prevent endless loops in the negotiation. It is required that a response be sent to requests for a change of mode -- even if the mode is not changed.

c. Whenever one party sends an option command to a second party, whether as a request or an acknowledgment, and use of the option will have any effect on the processing of the data being sent from the first party to the second, then the command must be inserted in the data stream at the point where it is desired that it take

effect. (It should be noted that some time will elapse between the transmission of a request and the receipt of an acknowledgment, which may be negative. Thus, a host may wish to

buffer data, after requesting an option, until it learns whether the request is accepted or rejected, in order to hide the "uncertainty period" from the user.)

Option requests are likely to flurry back and forth when a TELNET connection is first established, as each party attempts to get the best possible service from the other party. Beyond that, however, options can be used to dynamically modify the characteristics of the connection to suit changing local conditions. For example, the NVT, as will be explained later, uses a transmission discipline well suited to the many "line at a time" applications such as BASIC, but poorly suited to the many "character at a time" applications such as NLS. A server might elect to devote the extra processor overhead required for a "character at a time" discipline when it was suitable for the local process and would negotiate an appropriate option. However, rather than then being permanently burdened with the extra processing overhead, it could switch (i.e., negotiate) back to NVT when the detailed control was no longer necessary.

It is possible for requests initiated by processes to stimulate a nonterminating request loop if the process responds to a rejection by merely re-requesting the option. To prevent such loops from occurring, rejected requests should not be repeated until something changes. Operationally, this can mean the process is running a different program, or the user has given another command, or whatever makes sense in the context of the given process and the given option. A good rule of thumb is that a re-request should only occur as a result of subsequent information from the other end of the connection or when demanded by local human intervention.

Option designers should not feel constrained by the somewhat limited syntax available for option negotiation. The intent of the simple syntax is to make it easy to have options -- since it is correspondingly easy to profess ignorance about them. If some particular option requires a richer negotiation structure than possible within "DO, DON'T, WILL, WON'T", the proper tack is to use "DO, DON'T, WILL, WON'T" to establish that both parties understand the option, and once this is accomplished a more exotic syntax can be used freely. For example, a party might send a request to alter (establish) line length. If it is accepted, then a different syntax can be used for actually negotiating the line length -- such a "sub-negotiation" might include fields for minimum allowable, maximum allowable and desired line lengths. The important concept is that

such expanded negotiations should never begin until some prior (standard) negotiation has established that both parties are capable of parsing the expanded syntax.

In summary, WILL XXX is sent, by either party, to indicate that party's desire (offer) to begin performing option XXX, DO XXX and

DON'T XXX being its positive and negative acknowledgments; similarly, DO XXX is sent to indicate a desire (request) that the other party (i.e., the recipient of the DO) begin performing option XXX, WILL XXX and WON'T XXX being the positive and negative acknowledgments. Since the NVT is what is left when no options are enabled, the DON'T and WON'T responses are guaranteed to leave the connection in a state which both ends can handle. Thus, all hosts may implement their TELNET processes to be totally unaware of options that are not supported, simply returning a rejection to (i.e., refusing) any option request that cannot be understood.

As much as possible, the TELNET protocol has been made server-user symmetrical so that it easily and naturally covers the user-user (linking) and server-server (cooperating processes) cases. It is hoped, but not absolutely required, that options will further this intent. In any case, it is explicitly acknowledged that symmetry is an operating principle rather than an ironclad rule.

A companion document, "TELNET Option Specifications," should be consulted for information about the procedure for establishing new options.

THE NETWORK VIRTUAL TERMINAL

The Network Virtual Terminal (NVT) is a bi-directional character device. The NVT has a printer and a keyboard. The printer responds to incoming data and the keyboard produces outgoing data which is sent over the TELNET connection and, if "echoes" are desired, to the NVT's printer as well. "Echoes" will not be expected to traverse the network (although options exist to enable a "remote" echoing mode of operation, no host is required to implement this option). The code set is seven-bit USASCII in an eight-bit field, except as modified herein. Any code conversion and timing considerations are local problems and do not affect the NVT.

TRANSMISSION OF DATA

Although a TELNET connection through the network is intrinsically full duplex, the NVT is to be viewed as a half-duplex device operating in a line-buffered mode. That is, unless and until

options are negotiated to the contrary, the following default conditions pertain to the transmission of data over the TELNET connection:

1) Insofar as the availability of local buffer space permits, data should be accumulated in the host where it is generated until a complete line of data is ready for transmission, or until some locally-defined explicit signal to transmit occurs. This signal could be generated either by a process or by a

human user.

The motivation for this rule is the high cost, to some hosts, of processing network input interrupts, coupled with the default NVT specification that "echoes" do not traverse the network. Thus, it is reasonable to buffer some amount of data at its source. Many systems take some processing action at the end of each input line (even line printers or card punches frequently tend to work this way), so the transmission should be triggered at the end of a line. On the other hand, a user or process may sometimes find it necessary or desirable to provide data which does not terminate at the end of a line; therefore implementers are cautioned to provide methods of locally signaling that all buffered data should be transmitted immediately.

2) When a process has completed sending data to an NVT printer and has no queued input from the NVT keyboard for further processing (i.e., when a process at one end of a TELNET connection cannot proceed without input from the other end), the process must transmit the TELNET Go Ahead (GA) command.

This rule is not intended to require that the TELNET GA command be sent from a terminal at the end of each line, since server hosts do not normally require a special signal (in addition to end-of-line or other locally-defined characters) in order to commence processing. Rather, the TELNET GA is designed to help a user's local host operate a physically half duplex terminal which has a "lockable" keyboard such as the IBM 2741. A description of this type of terminal may help to explain the proper use of the GA command.

The terminal-computer connection is always under control of either the user or the computer. Neither can unilaterally seize control from the other; rather the controlling end must relinguish its control explicitly. At the terminal end, the hardware is constructed so as to relinquish control each time that a "line" is terminated (i.e., when the "New Line" key is typed by the user). When this occurs, the attached (local)

computer processes the input data, decides if output should be generated, and if not returns control to the terminal. If output should be generated, control is retained by the computer until all output has been transmitted.

The difficulties of using this type of terminal through the network should be obvious. The "local" computer is no longer able to decide whether to retain control after seeing an end-of-line signal or not; this decision can only be made by the "remote" computer which is processing the data. Therefore, the TELNET GA command provides a mechanism whereby the "remote" (server) computer can signal the "local" (user) computer that

it is time to pass control to the user of the terminal. It should be transmitted at those times, and only at those times, when the user should be given control of the terminal. Note that premature transmission of the GA command may result in the blocking of output, since the user is likely to assume that the transmitting system has paused, and therefore he will fail to turn the line around manually.

The foregoing, of course, does not apply to the user-to-server direction of communication. In this direction, GAs may be sent at any time, but need not ever be sent. Also, if the TELNET connection is being used for process-to-process communication, GAs need not be sent in either direction. Finally, for terminal-to-terminal communication, GAs may be required in neither, one, or both directions. If a host plans to support terminal-to-terminal communication it is suggested that the host provide the user with a means of manually signaling that it is time for a GA to be sent over the TELNET connection; this, however, is not a requirement on the implementer of a TELNET process.

Note that the symmetry of the TELNET model requires that there is an NVT at each end of the TELNET connection, at least conceptually.

STANDARD REPRESENTATION OF CONTROL FUNCTIONS

As stated in the Introduction to this document, the primary goal of the TELNET protocol is the provision of a standard interfacing of terminal devices and terminal-oriented processes through the network. Early experiences with this type of interconnection have shown that certain functions are implemented by most servers, but that the methods of invoking these functions differ widely. For a human user who interacts with several server systems, these differences are highly frustrating. TELNET, therefore, defines a standard representation for five of these functions, as described

below. These standard representations have standard, but not required, meanings (with the exception that the Interrupt Process (IP) function may be required by other protocols which use TELNET); that is, a system which does not provide the function to local users need not provide it to network users and may treat the standard representation for the function as a No-operation. On the other hand, a system which does provide the function to a local user is obliged to provide the same function to a network user who transmits the standard representation for the function.

Interrupt Process (IP)

Many systems provide a function which suspends, interrupts, aborts, or terminates the operation of a user process. This function is frequently used when a user believes his process is

in an unending loop, or when an unwanted process has been inadvertently activated. IP is the standard representation for invoking this function. It should be noted by implementers that IP may be required by other protocols which use TELNET, and therefore should be implemented if these other protocols are to be supported.

Abort Output (AO)

Many systems provide a function which allows a process, which is generating output, to run to completion (or to reach the same stopping point it would reach if running to completion) but without sending the output to the user's terminal. Further, this function typically clears any output already produced but not yet actually printed (or displayed) on the user's terminal. AO is the standard representation for invoking this function. For example, some subsystem might normally accept a user's command, send a long text string to the user's terminal in response, and finally signal readiness to accept the next command by sending a "prompt" character (preceded by <CR><LF>) to the user's terminal. If the AO were received during the transmission of the text string, a reasonable implementation would be to suppress the remainder of the text string, but transmit the prompt character and the preceding <CR><LF>. (This is possibly in distinction to the action which might be taken if an IP were received; the IP might cause suppression of the text string and an exit from the subsystem.)

It should be noted, by server systems which provide this function, that there may be buffers external to the system (in

the network and the user's local host) which should be cleared; the appropriate way to do this is to transmit the "Synch" signal (described below) to the user system.

Are You There (AYT)

Many systems provide a function which provides the user with some visible (e.g., printable) evidence that the system is still up and running. This function may be invoked by the user when the system is unexpectedly "silent" for a long time, because of the unanticipated (by the user) length of a computation, an unusually heavy system load, etc. AYT is the standard representation for invoking this function.

Erase Character (EC)

Many systems provide a function which deletes the last preceding undeleted character or "print position"* from the

stream of data being supplied by the user. This function is typically used to edit keyboard input when typing mistakes are made. EC is the standard representation for invoking this function.

*NOTE: A "print position" may contain several characters which are the result of overstrikes, or of sequences such as <char1> BS <char2>...

Erase Line (EL)

Many systems provide a function which deletes all the data in the current "line" of input. This function is typically used to edit keyboard input. EL is the standard representation for invoking this function.

THE TELNET "SYNCH" SIGNAL

Most time-sharing systems provide mechanisms which allow a terminal user to regain control of a "runaway" process; the IP and AO functions described above are examples of these mechanisms. Such systems, when used locally, have access to all of the signals supplied by the user, whether these are normal characters or special "out of band" signals such as those supplied by the teletype "BREAK" key or the IBM 2741 "ATTN" key. This is not necessarily true when terminals are connected to the system through the network; the network's flow control mechanisms may cause such a signal to be buffered elsewhere, for example in the user's host.

To counter this problem, the TELNET "Synch" mechanism is introduced. A Synch signal consists of a TCP Urgent notification, coupled with the TELNET command DATA MARK. The Urgent notification, which is not subject to the flow control pertaining to the TELNET connection, is used to invoke special handling of the data stream by the process which receives it. In this mode, the data stream is immediately scanned for "interesting" signals as defined below, discarding intervening data. The TELNET command DATA MARK (DM) is the synchronizing mark in the data stream which indicates that any special signal has already occurred and the recipient can return to normal processing of the data stream.

The Synch is sent via the TCP send operation with the Urgent flag set and the DM as the last (or only) data octet.

When several Synchs are sent in rapid succession, the Urgent notifications may be merged. It is not possible to count Urgents since the number received will be less than or equal the number sent. When in normal mode, a DM is a no operation; when in urgent mode, it signals the end of the urgent processing.

If TCP indicates the end of Urgent data before the DM is found, TELNET should continue the special handling of the data stream until the DM is found.

If TCP indicates more Urgent data after the DM is found, it can only be because of a subsequent Synch. TELNET should continue the special handling of the data stream until another DM is found.

"Interesting" signals are defined to be: the TELNET standard representations of IP, AO, and AYT (but not EC or EL); the local analogs of these standard representations (if any); all other TELNET commands; other site-defined signals which can be acted on without delaying the scan of the data stream.

Since one effect of the SYNCH mechanism is the discarding of essentially all characters (except TELNET commands) between the sender of the Synch and its recipient, this mechanism is specified as the standard way to clear the data path when that is desired. For example, if a user at a terminal causes an AO to be transmitted, the server which receives the AO (if it provides that function at all) should return a Synch to the user.

Finally, just as the TCP Urgent notification is needed at the TELNET level as an out-of-band signal, so other protocols which make use of TELNET may require a TELNET command which can be viewed as an out-of-band signal at a different level.

By convention the sequence [IP, Synch] is to be used as such a signal. For example, suppose that some other protocol, which uses TELNET, defines the character string STOP analogously to the TELNET command AO. Imagine that a user of this protocol wishes a server to process the STOP string, but the connection is blocked because the server is processing other commands. The user should instruct his system to:

1. Send the TELNET IP character;

2. Send the TELNET SYNC sequence, that is:

   Send the Data Mark (DM) as the only character in a TCP urgent mode send operation.

3. Send the character string STOP; and

4. Send the other protocol's analog of the TELNET DM, if any.

The user (or process acting on his behalf) must transmit the TELNET SYNCH sequence of step 2 above to ensure that the TELNET IP gets through to the server's TELNET interpreter.

The Urgent should wake up the TELNET process; the IP should

wake up the next higher level process.

THE NVT PRINTER AND KEYBOARD

The NVT printer has an unspecified carriage width and page length and can produce representations of all 95 USASCII graphics (codes 32 through 126). Of the 33 USASCII control codes (0 through 31 and 127), and the 128 uncovered codes (128 through 255), the following have specified meaning to the NVT printer:

NAME	CODE	MEANING
NULL (NUL)	0	No Operation
Line Feed (LF)	10	Moves the printer to the next print line, keeping the same horizontal position.
Carriage Return (CR)	13	Moves the printer to the left margin of the current line.

In addition, the following codes shall have defined, but not required, effects on the NVT printer. Neither end of a TELNET connection may assume that the other party will take, or will have taken, any particular action upon receipt or transmission of these:

BELL (BEL)	7	Produces an audible or visible signal (which does NOT move the print head).
Back Space (BS)	8	Moves the print head one character position towards the left margin.
Horizontal Tab (HT)	9	Moves the printer to the next horizontal tab stop. It remains unspecified how either party determines or establishes where such tab stops are located.
Vertical Tab (VT)	11	Moves the printer to the next vertical tab stop. It remains unspecified how either party determines or establishes where such tab stops are located.
Form Feed (FF)	12	Moves the printer to the top of the next page, keeping the same horizontal position.

All remaining codes do not cause the NVT printer to take any action.

The sequence "CR LF", as defined, will cause the NVT to be positioned at the left margin of the next print line (as would, for example, the sequence "LF CR"). However, many systems and terminals do not treat CR and LF independently, and will have to go to some effort to simulate their effect. (For example, some terminals do not have a CR independent of the LF, but on such terminals it may be possible to simulate a CR by backspacing.) Therefore, the sequence "CR LF" must be treated as a single "new line" character and used whenever their combined action is intended; the sequence "CR NUL" must be used where a carriage return alone is actually desired; and the CR character must be avoided in other contexts. This rule gives assurance to systems which must decide whether to perform a "new line" function or a multiple-backspace that the TELNET stream contains a character following a CR that will allow a rational decision.

Note that "CR LF" or "CR NUL" is required in both directions

(in the default ASCII mode), to preserve the symmetry of the NVT model. Even though it may be known in some situations (e.g., with remote echo and suppress go ahead options in effect) that characters are not being sent to an actual printer, nonetheless, for the sake of consistency, the protocol requires that a NUL be inserted following a CR not followed by a LF in the data stream. The converse of this is that a NUL received in the data stream after a CR (in the absence of options negotiations which explicitly specify otherwise) should be stripped out prior to applying the NVT to local character set mapping.

The NVT keyboard has keys, or key combinations, or key sequences, for generating all 128 USASCII codes. Note that although many have no effect on the NVT printer, the NVT keyboard is capable of generating them.

In addition to these codes, the NVT keyboard shall be capable of generating the following additional codes which, except as noted, have defined, but not reguired, meanings. The actual code assignments for these "characters" are in the TELNET Command section, because they are viewed as being, in some sense, generic and should be available even when the data stream is interpreted as being some other character set.

Synch

This key allows the user to clear his data path to the other party. The activation of this key causes a DM (see command section) to be sent in the data stream and a TCP Urgent

notification is associated with it. The pair DM-Urgent is to have required meaning as defined previously.

Break (BRK)

This code is provided because it is a signal outside the USASCII set which is currently given local meaning within many systems. It is intended to indicate that the Break Key or the Attention Key was hit. Note, however, that this is intended to provide a 129th code for systems which require it, not as a synonym for the IP standard representation.

Interrupt Process (IP)

Suspend, interrupt, abort or terminate the process to which the NVT is connected. Also, part of the out-of-band signal for other protocols which use TELNET.

Abort Output (AO)

Allow the current process to (appear to) run to completion, but do not send its output to the user. Also, send a Synch to the user.

Are You There (AYT)

Send back to the NVT some visible (i.e., printable) evidence that the AYT was received.

Erase Character (EC)

The recipient should delete the last preceding undeleted character or "print position" from the data stream.

Erase Line (EL)

The recipient should delete characters from the data stream back to, but not including, the last "CR LF" sequence sent over the TELNET connection.

The spirit of these "extra" keys, and also the printer format effectors, is that they should represent a natural extension of the mapping that already must be done from "NVT" into "local". Just as the NVT data byte 68 (104 octal) should be mapped into whatever the local code for "uppercase D" is, so the EC character should be mapped into whatever the local "Erase Character" function is. Further, just as the mapping for 124 (174 octal) is somewhat arbitrary in an environment that has no "vertical bar" character, the EL character may have a somewhat arbitrary mapping (or none at all) if there is no local "Erase Line" facility. Similarly for format effectors: if the terminal actually does

have a "Vertical Tab", then the mapping for VT is obvious, and only when the terminal does not have a vertical tab should the effect of VT be unpredictable.

TELNET COMMAND STRUCTURE

All TELNET commands consist of at least a two byte sequence: the "Interpret as Command" (IAC) escape character followed by the code for the command. The commands dealing with option negotiation are three byte sequences, the third byte being the code for the option referenced. This format was chosen so that as more comprehensive use of the "data space" is made -- by negotiations from the basic NVT, of course -- collisions of data bytes with reserved command values will be minimized, all such collisions requiring the inconvenience, and

inefficiency, of "escaping" the data bytes into the stream. With the current set-up, only the IAC need be doubled to be sent as data, and the other 255 codes may be passed transparently.

The following are the defined TELNET commands. Note that these codes and code sequences have the indicated meaning only when immediately preceded by an IAC.

NAME	CODE	MEANING
SE	240	End of subnegotiation parameters.
NOP	241	No operation.
Data Mark	242	The data stream portion of a Synch. This should always be accompanied by a TCP Urgent notification.
Break	243	NVT character BRK.
Interrupt Process	244	The function IP.
Abort output	245	The function AO.
Are You There	246	The function AYT.
Erase character	247	The function EC.
Erase Line	248	The function EL.
Go ahead	249	The GA signal.
SB	250	Indicates that what follows is subnegotiation of the indicated option.
WILL (option code)	251	Indicates the desire to begin performing, or confirmation that you are now performing, the indicated option.
WON'T (option code)	252	Indicates the refusal to perform, or continue performing, the indicated option.
DO (option code)	253	Indicates the request that the other party perform, or confirmation that you are expecting the other party to perform, the

		indicated option.
DON'T (option code)	254	Indicates the demand that the other party stop performing, or confirmation that you are no longer expecting the other party to perform, the indicated option.
IAC	255	Data Byte 255.

CONNECTION ESTABLISHMENT

The TELNET TCP connection is established between the user's port U and the server's port L. The server listens on its well known port L for such connections. Since a TCP connection is full duplex and identified by the pair of ports, the server can engage in many simultaneous connections involving its port L and different user ports U.

Port Assignment

When used for remote user access to service hosts (i.e., remote terminal access) this protocol is assigned server port 23 (27 octal). That is L=23.

Network Working Group
Request for Comments: 792

J. Postel
ISI
September 1981

Updates: RFCs 777, 760
Updates: IENs 109, 128

INTERNET CONTROL MESSAGE PROTOCOL

DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION

Introduction

The Internet Protocol (IP) [1] is used for host-to-host datagram service in a system of interconnected networks called the Catenet [2]. The network connecting devices are called Gateways. These gateways communicate between themselves for control purposes via a Gateway to Gateway Protocol (GGP) [3,4]. Occasionally a gateway or destination host will communicate with a source host, for example, to report an error in datagram processing. For such purposes this protocol, the Internet Control Message Protocol (ICMP), is used. ICMP, uses the basic support of IP as if it were a higher level protocol, however, ICMP is actually an integral part of IP, and must be implemented by every IP module.

ICMP messages are sent in several situations: for example, when a datagram cannot reach its destination, when the gateway does not have the buffering capacity to forward a datagram, and when the gateway can direct the host to send traffic on a shorter route.

The Internet Protocol is not designed to be absolutely reliable. The purpose of these control messages is to provide feedback about problems in the communication environment, not to make IP reliable. There are still no guarantees that a datagram will be delivered or a control message will be returned. Some datagrams may still be undelivered without any report of their loss. The higher level protocols that use IP must implement their own reliability procedures if reliable communication is required.

The ICMP messages typically report errors in the processing of datagrams. To avoid the infinite regress of messages about messages etc., no ICMP messages are sent about ICMP messages. Also ICMP messages are only sent about errors in handling fragment zero of fragemented datagrams. (Fragment zero has the fragment offeset equal zero).

Message Formats

ICMP messages are sent using the basic IP header. The first octet of the data portion of the datagram is a ICMP type field; the value of this field determines the format of the remaining data. Any field labeled "unused" is reserved for later extensions and must be zero when sent, but receivers should not use these fields (except to include them in the checksum). Unless otherwise noted under the individual format descriptions, the values of the internet header fields are as follows:

Version

4

IHL

Internet header length in 32-bit words.

Type of Service

0

Total Length

Length of internet header and data in octets.

Identification, Flags, Fragment Offset

Used in fragmentation, see [1].

Time to Live

Time to live in seconds; as this field is decremented at each machine in which the datagram is processed, the value in this field should be at least as great as the number of gateways which this datagram will traverse.

Protocol

ICMP = 1

Header Checksum

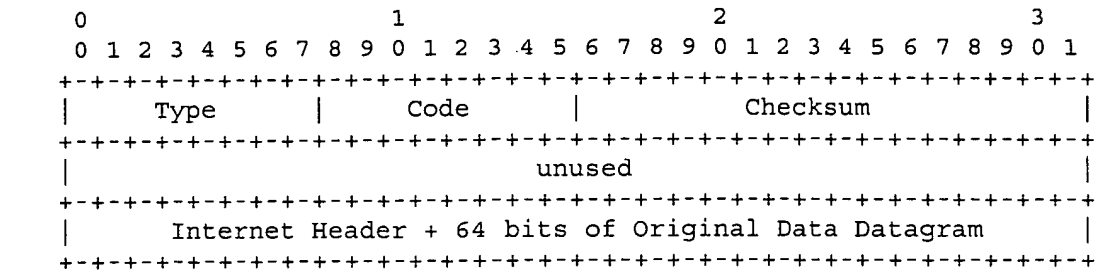
The 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Source Address

The address of the gateway or host that composes the ICMP message. Unless otherwise noted, this can be any of a gateway's addresses.

Destination Address

The address of the gateway or host to which the message should be sent.



IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

3

Code

0 = net unreachable;

1 = host unreachable;

2 = protocol unreachable;

3 = port unreachable;

4 = fragmentation needed and DF set;

5 = source route failed.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original

datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

If, according to the information in the gateway's routing tables, the network specified in the internet destination field of a datagram is unreachable, e.g., the distance to the network is infinity, the gateway may send a destination unreachable message to the internet source host of the datagram. In addition, in some networks, the gateway may be able to determine if the internet destination host is unreachable. Gateways in these networks may send destination unreachable messages to the source host when the destination host is unreachable.

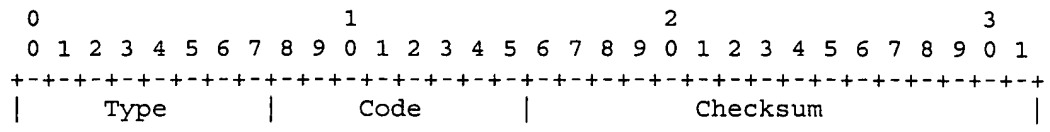
If, in the destination host, the IP module cannot deliver the datagram because the indicated protocol module or process port is not active, the destination host may send a destination unreachable message to the source host.

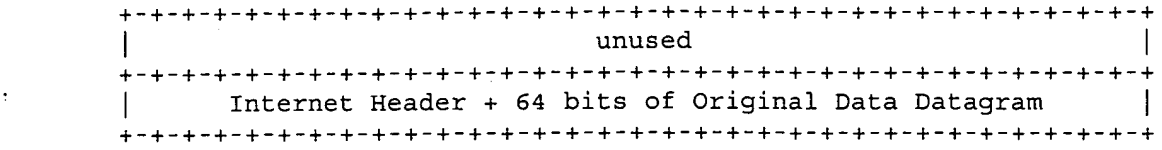
Another case is when a datagram must be fragmented to be forwarded by a gateway yet the Don't Fragment flag is on. In this case the gateway must discard the datagram and may return a destination unreachable message.

Codes 0, 1, 4, and 5 may be received from a gateway. Codes 2 and 3 may be received from a host.

□

Time Exceeded Message





IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

11

Code

0 = time to live exceeded in transit;

1 = fragment reassembly time exceeded.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

If the gateway processing a datagram finds the time to live field

is zero it must discard the datagram. The gateway may also notify the source host via the time exceeded message.

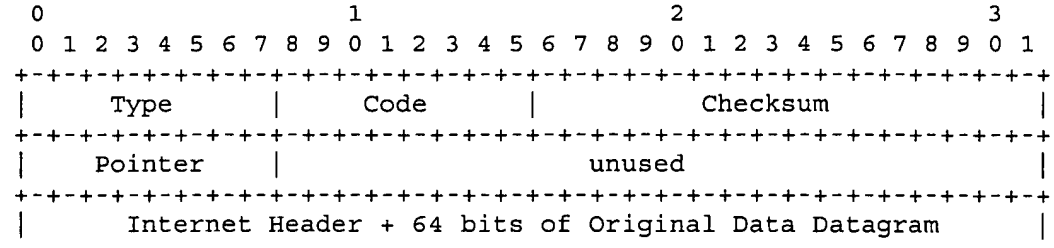
If a host reassembling a fragmented datagram cannot complete the reassembly due to missing fragments within its time limit it discards the datagram, and it may send a time exceeded message.

If fragment zero is not available then no time exceeded need be

sent at all.

Code 0 may be received from a gateway. Code 1 may be received from a host.

Parameter Problem Message



```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

12

Code

0 = pointer indicates the error.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Pointer

If code = 0, identifies the octet where an error was detected.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

If the gateway or host processing a datagram finds a problem with the header parameters such that it cannot complete processing the datagram it must discard the datagram. One potential source of such a problem is with incorrect arguments in an option. The gateway or host may also notify the source host via the parameter problem message. This message is only sent if the error caused the datagram to be discarded.

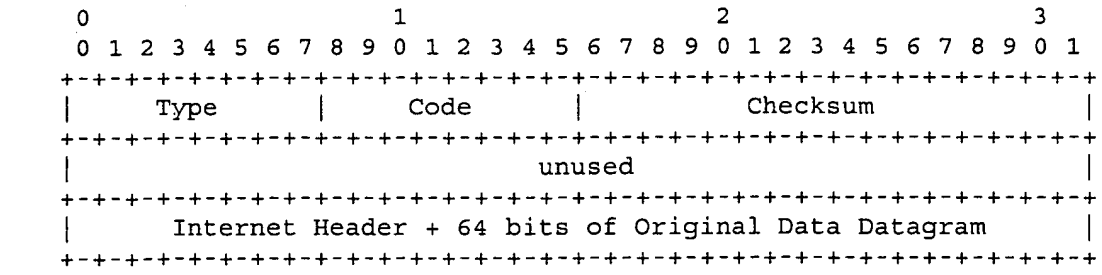
The pointer identifies the octet of the original datagram's header where the error was detected (it may be in the middle of an

option). For example, 1 indicates something is wrong with the Type of Service, and (if there are options present) 20 indicates something is wrong with the type code of the first option.

Code 0 may be received from a gateway or a host.

□

Source Quench Message



IP Fields:

Destination Address

The source network and address of the original datagram's data.

ICMP Fields:

Type

4

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

A gateway may discard internet datagrams if it does not have the buffer space needed to queue the datagrams for output to the next network on the route to the destination network. If a gateway

discards a datagram, it may send a source quench message to the internet source host of the datagram. A destination host may also send a source quench message if datagrams arrive too fast to be processed. The source quench message is a request to the host to cut back the rate at which it is sending traffic to the internet destination. The gateway may send a source quench message for every message that it discards. On receipt of a source quench message, the source host should cut back the rate at which it is sending traffic to the specified destination until it no longer receives source quench messages from the gateway. The source host can then gradually increase the rate at which it sends traffic to the destination until it again receives source quench messages.

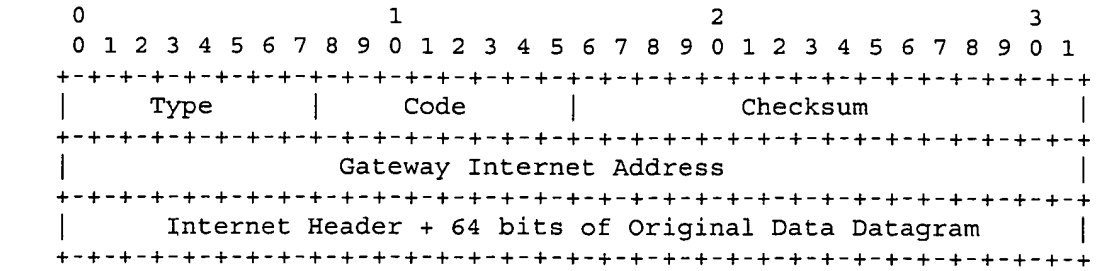
The gateway or host may send the source quench message when it approaches its capacity limit rather than waiting until the capacity is exceeded. This means that the data datagram which

triggered the source quench message may be delivered.

Code 0 may be received from a gateway or a host.

□

Redirect Message



IP Fields:

Destination Address

The source network and address of the original datagram's data.

ICMP Fields:

Type

5

Code

0 = Redirect datagrams for the Network.

1 = Redirect datagrams for the Host.

2 = Redirect datagrams for the Type of Service and Network.

3 = Redirect datagrams for the Type of Service and Host.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Gateway Internet Address

Address of the gateway to which traffic for the network specified in the internet destination network field of the original datagram's data should be sent.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

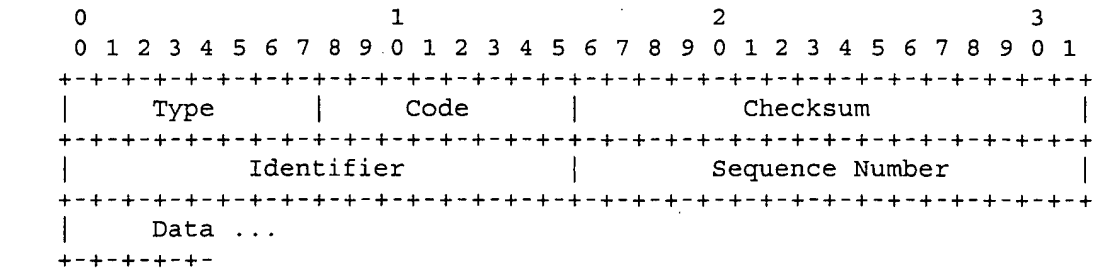
The gateway sends a redirect message to a host in the following situation. A gateway, G1, receives an internet datagram from a host on a network to which the gateway is attached. The gateway, G1, checks its routing table and obtains the address of the next gateway, G2, on the route to the datagram's internet destination network, X. If G2 and the host identified by the internet source address of the datagram are on the same network, a redirect message is sent to the host. The redirect message advises the host to send its traffic for network X directly to gateway G2 as this is a shorter path to the destination. The gateway forwards

the original datagram's data to its internet destination.

For datagrams with the IP source route options and the gateway address in the destination address field, a redirect message is not sent even if there is a better route to the ultimate destination than the next address in the source route.

Codes 0, 1, 2, and 3 may be received from a gateway.

Echo or Echo Reply Message



IP Fields:

Addresses

The address of the source in an echo message will be the destination of the echo reply message. To form an echo reply message, the source and destination addresses are simply reversed, the type code changed to 0, and the checksum recomputed.

IP Fields:

Type

8 for echo message;

0 for echo reply message.

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. If the total length is odd, the received data is padded with one octet of zeros for computing the checksum. This checksum may be replaced in the future.

Identifier

If code = 0, an identifier to aid in matching echos and replies, may be zero.

Sequence Number

If code = 0, a sequence number to aid in matching echos and replies, may be zero.

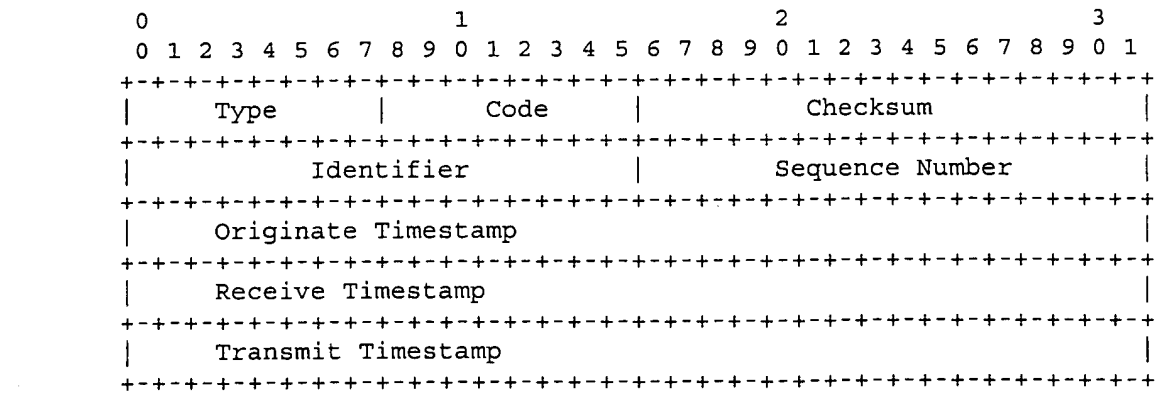
Description

The data received in the echo message must be returned in the echo reply message.

The identifier and sequence number may be used by the echo sender to aid in matching the replies with the echo requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each echo request sent. The echoer returns these same values in the echo reply.

Code 0 may be received from a gateway or a host.

Timestamp or Timestamp Reply Message



IP Fields:

Addresses

The address of the source in a timestamp message will be the destination of the timestamp reply message. To form a timestamp reply message, the source and destination addresses are simply reversed, the type code changed to 14, and the checksum recomputed.

IP Fields:

Type

13 for timestamp message;

14 for timestamp reply message.

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Identifier

If code = 0, an identifier to aid in matching timestamp and replies, may be zero.

Sequence Number

If code = 0, a sequence number to aid in matching timestamp and replies, may be zero.

Description

The data received (a timestamp) in the message is returned in the reply together with an additional timestamp. The timestamp is 32 bits of milliseconds since midnight UT. One use of these timestamps is described by Mills [5].

The Originate Timestamp is the time the sender last touched the message before sending it, the Receive Timestamp is the time the echoer first touched it on receipt, and the Transmit Timestamp is the time the echoer last touched the message on sending it.

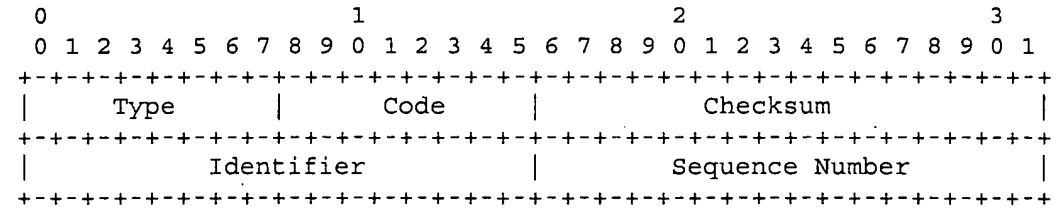
If the time is not available in miliseconds or cannot be provided with respect to midnight UT then any time can be inserted in a timestamp provided the high order bit of the timestamp is also set to indicate this non-standard value.

The identifier and sequence number may be used by the echo sender to aid in matching the replies with the requests. For example, the identifier might be used like a port in TCP or UDP to identify

a session, and the sequence number might be incremented on each request sent. The destination returns these same values in the reply.

Code 0 may be received from a gateway or a host.

Information Request or Information Reply Message



IP Fields:

Addresses

The address of the source in a information request message will be the destination of the information reply message. To form a information reply message, the source and destination addresses are simply reversed, the type code changed to 16, and the checksum recomputed.

IP Fields:

Type

15 for information request message;

16 for information reply message.

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Identifier

If code = 0, an identifier to aid in matching request and replies, may be zero.

Sequence Number

If code = 0, a sequence number to aid in matching request and replies, may be zero.

Description

This message may be sent with the source network in the IP header source and destination address fields zero (which means "this" network). The replying IP module should send the reply with the addresses fully specified. This message is a way for a host to find out the number of the network it is on.

The identifier and sequence number may be used by the echo sender to aid in matching the replies with the requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each request sent. The destination returns these same values in the reply.

Code 0 may be received from a gateway or a host.

Summary of Message Types

0	Echo Reply
3	Destination Unreachable
4	Source Quench
5	Redirect
8	Echo
11	Time Exceeded
12	Parameter Problem
13	Timestamp
14	Timestamp Reply
15	Information Request
16	Information Reply

References

[1] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification," RFC 791, USC/Information Sciences Institute, September 1981.

[2] Cerf, V., "The Catenet Model for Internetworking," IEN 48, Information Processing Techniques Office, Defense Advanced Research Projects Agency, July 1978.

[3] Strazisar, V., "Gateway Routing: An Implementation Specification", IEN 30, Bolt Beranek and Newman, April 1979.

[4] Strazisar, V., "How to Build a Gateway", IEN 109, Bolt Beranek and Newman, August 1979.

[5] Mills, D., "DCNET Internet Clock Service," RFC 778, COMSAT Laboratories, April 1981.